ECE482 — Introduction to Operating Systems

Lab 8

Manuel — JI (Fall 2024)

Goals of the lab

- Understand the basics about memory
- Compare two page replacement algorithms
- Work with Minix 3 kernel source code

ECE4820 Tasks

1 Introduction

You have handed in your `mumsh` a long time ago, but your mum is still complaining about your work: "You were too slow! and you almost missed some milestones", she said, "If you keep being this slow, you will get fired by Lemonion Inc.." You immediately try to explain that Mr. Frown is making your life a nightmare, but she cuts you short: "Stop making up stories to excuse your slowness and bad work quality! I met Mr. Frown over a week ago when I came to bring your lunch box that you had forgotten and he is a real gentlemen. He was very kind, he even offered me a coffee and a piece of cake. We had a friendly conversation and to be honest he was not surprised to hear that you were spending all your time playing video games... He comforted me, saying that he subdued worse kids than you and you will soon be back on track, he would personally take care of you! I cannot believe a single word of all the mean things you say about that selfless man. You should be ashamed of slandering such a nice person, and instead should be thankful for everything he does for you. This is not how I have raised you!"

Your mum is obviously very upset at you, and on those last words she leaves you alone with your thoughts. This double face Mr. Frown managed to even ruin your family time. How can someone be so evil? You have no idea how much longer you will be able to cope with this situation. It was still acceptable when you were in peace at home, but now with mum on your back that really feels terrible. In any case you now understand why things are getting worse and worse with Mr. Frown; that was not a mere feeling, he really is meaner than ever with you!

As you ponder over what has just happened you find so many arguments why mum is wrong about you being too slow with `mumsh`. First you had very little knowledge about shell and terminal when she assigned you the task, then C is a language you learned many years ago and almost never used since then. In fact this is your Least Recently Used (LRU) language, so it makes sense that you have forgotten most of it.

But at the same time it felt like Mum Genuinely Looked Really Upset (MGLRU). As this acronym sounds familiar you check online and discover this article about Multi-Gen LRU. Following that reading you feel inspired, what if you could monitor the page fault rate with and without MGLRU?

2 Looking at the Linux kernel

Right away you decide to jump into the Linux kernel source code and figure out how Memory Management (MM) works. You quickly notice that things are extremely complicated, but this makes sense as MM is one of the lowest part of the kernel where much interaction with the hardware is necessary. Therefore

you decide to only get a high level perspective on the content, i.e. you look at files mostly trying to understand the underlying logic based on the tree structure, the source code comments, and the function names.

First you notice references to concepts and topics you have already seen but never dared to look at in details. The all seem to be related to coding quality, which you already know is of a critical importance when working at the kernel level.

- What are `kmsan, kasan, kfence` folders?
- What is *shadow memory* and how does it relate to sanitizers?
- What are sanitizers and why are they essential when coding in the kernel level?
- What is `kmemleak.c` file about? Explain who should "use" it and under what circumstances.

You also notice two interesting files:

- `nommu.c`: when would this file be used? List some drawbacks of having CPU without any MMU.
- `compaction.c`: what is this file used for? Why is "memory compaction" important?

As you keep looking at the files in the `mm` directory of the Linux source code as name stands out: `oom_kill.c`. This sounds scary! What could be killed. You open the files and discover that it features long interesting comments.

- What is the OOM killer? In particular what does OOM mean?
- Based on the source code and comments, explain what could be a reason for a deadlock in the Linux kernel? *Hint.* There is no need to read the whole source code, simply run a search on a well chosen keyword.

As you keep exploring the `mm` directory you notice a few swap related filenames. Without opening them you start thinking of the swap and a few questions come to your mind. You know that the kernel and user spaces memory are separated but what about the swap? You decide to run a quick online search to understand

- the "rough" layout of kernel and user space memories;
- whether or not kernel memory can be swapped and why;

At this stage it is pretty clear that directly working in the kernel for MM is to complicated. Unfortunately with the pressure Mr. Frown puts on you, it is very hard for you to find enough time to properly investigate the benefits of MGLRU as you initially intended. Therefore you decide to keep searching online for some task that would prove that Mum Generally Loves to be Really Unfair! And again you are back with this MGLRU! Interestingly you suddenly land on this kernel documentation page. It shows how to enable and disable MGLRU from the user-space! What if you could work with MM from the user-space?

2.1 A brief introduction to eBPF

While searching information about MM in user-space you notice many recent research articles related to something called eBPF. As this seems to be a brand new topic this is likely a very idea to look into it: maybe being knowledgeable on it could earn you a better job without Mr. Frown! This really looks worth the effort, so you decide to read some documentation and follow a short presentation on the topic.

2.2 A page-fault handler using eBPF

At the end of the presentation you realise that you have everything you need! The eBPF approach can work magic for you: no need to dig any further into Linux MM with long and complex code. You can simply run memory pressure tests with and without MGLRU enabled. For that you just need to monitor page-faults using an eBPF program. This sounds like a nice starting point to learn more about this intriguing technology which bridges the gap between monolithic and micro kernels, by safely "injecting" and "running" user-defined code into kernel space!

ECE4821 Tasks

2.3 More on MM

Now that you have a better idea of how important memory is in an OS and have a rough idea of how it is implemented, it is time for you to look at slightly more advanced things which would bets prepare you at more advanced Linux kernel coding.

- What is the difference between `vmalloc` and `kmalloc`? Which one is it often best to use?
- What is `/dev/shm`? Explain a simple use case.
- What is page poisoning and when could it be useful? *Hint.* Look at `page_poison.c`.
- Where is the page table structure defined? *Hint.* Where are architecture specific header files located and why can you expect to find information there?
- Why should the memory be handled differently at boot and run times? *Hint.* Check `memblock.c`.

Proud of yourself and your accomplishments to decide to quickly go to sleep as the night is already well advanced and you know that tomorrow early morning you are going to face an even worse Mr. Frown. As there is no point on focusing on such negative thoughts, you prefer to relish on your victory.